Minimum-sum dipolar spanning tree for points in $\mathbb{R}^3$

Steven Bitner*   Ovidiu Daescu†

Abstract

We discuss the problem of finding the minimum-sum dipolar spanning tree (MSST) in three dimensions. The MSST problem is a minimization problem wherein given a set S of n points the goal is to find two points $x, y \in S$ that minimize the sum $|xy| + \max\{r_x, r_y\}$, where r_x and r_y are the radii of two disks with centers at x and y, respectively, that together cover all points in S. We present an $O(n^2 \log^2 n)$ time algorithm that uses $O(n^2)$ space, improving upon the known three dimensional result of $O(n^{2.5+\epsilon})$ time and $O(n^2)$ space.

1 Introduction

For a set S of n points, the geometric minimum diameter spanning tree (MDST) is defined as a spanning tree of S that minimizes the Euclidean length of the longest path in the tree. In [6], it has been proven that there always exists a monopolar or a dipolar MDST, i.e., a MDST with only one or two nodes of degree greater than one. A monopolar MDST can be found in $O(n \log n)$ time [6]. For the dipolar MDST the goal is to find two points $x, y \in S$ that minimize the sum $r_x + |xy| + r_y$, where $|xy|$ is the Euclidean distance between the points x and y, and r_x and r_y are the radii of two disks with centers at x and y, respectively, that together cover all points in S. The best known result is based on semi-dynamic data structures and achieves $O^*(n^{3-c_d})$ time [2], where the O^*-notation hides an $o(n^\epsilon)$ term, for any constant $\epsilon > 0$, and $c_d = 1/((d+1)(\lfloor d/2 \rfloor + 1))$ is a constant that depends on the dimension d of the point set. For example, $c_2 = 1/6$ and $c_3 = 1/12$.

In [4] they introduce a related (facility location) problem, the *minimum-sum dipolar spanning tree* (MSST) problem, in which the goal is to find two points $x, y \in S$ that minimize the sum $|xy| + \max\{r_x, r_y\}$. They present exact results when S is a set of n points in $\mathbb{R}^d$, for $d \in \{2, 3, 4\}$. For the planar case, their algorithm takes $O(n^2 \log n)$ time using $O(n^2)$ space. For dimensions $d = \{3, 4\}$, they suggest a solution based on range searching that takes $O(n^{2.5+\epsilon})$ time using $O(n^2)$ space, for any constant $\epsilon > 0$.

Results. In this paper we consider finding a MSST in $\mathbb{R}^3$, and present an algorithm that takes $O(n^2 \log^2 n)$ time using $O(n^2)$ space, thus almost matching the best known results for the planar case. To achieve this, we prove an interesting result related to the complexity of the common intersection of n balls in $\mathbb{R}^3$, of possible different radii, that are all tangent to a given point p.

Definitions and terminology. For two points a and b, $|ab|$ denotes the Euclidean distance from a to b. We use $\Sigma(a, b)$ to denote the ball centered at a and having b on its bounding sphere, that is, the radius of the bounding sphere of $\Sigma(a, b)$ has length $|ab|$.

Let p, q be two points in S, and let Π be the plane that is the perpendicular bisector of the line segment $\overline{pq}$. We use h_{pq} to denote the open half-space bounded by Π and containing p. Similarly, h_{qp} denotes the open half-space bounded by Π and containing q.

Given $p, q \in S$, the q-farthest point f_{pq} is defined as the farthest point from p that is contained in the open halfspace h_{pq} (see Fig. 1). A critical step in our solution is finding f_{pq} for a fixed p and all $q \in S \setminus \{p\}$ efficiently.

*Department of Computer Science, University of Texas at Dallas, stevenbitner@student.utdallas.edu

†Department of Computer Science, University of Texas at Dallas, daescu@utdallas.edu Daescu's research was partially supported by NSF grant CCF-0635013.

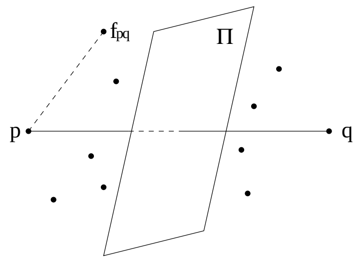

Figure 1: p, q, and the q-farthest point f_{pq}.

2 Finding the MSST in $\mathbb{R}^3$

In this section we present our solution for finding the MSST in $\mathbb{R}^3$. To this end, we extend to $\mathbb{R}^3$ a lemma from [4] (Lemma 1 below), prove a key property on the complexity of the common intersection of balls all tangent to a point p, and give an algorithm to compute the MSST within the claimed time and space bounds.

Lemma 1 *The point $x \in S$ is the q-farthest point from p iff x is the farthest point from p satisfying $q \notin \Sigma(x,p)$.*

Proof. $\Rightarrow$ Since x is the q-farthest point from p, by definition, it is contained in the open half-space h_{pq} and no other point of S in h_{pq} is farther from p than x. Note that all points of $S \cap h_{qp}$ must have a smaller distance to q than to p since the halfspace h_{qp} is defined by the orthogonal bisecting plane of $\overline{pq}$ (see Fig. 1). Then, $q \in \Sigma(x,p)$ would imply $|xq| < |xp|$, which means $x \in h_{qp}$, a contradiction. Thus, $q \notin \Sigma(x,p)$.

$\Leftarrow$ Since $q \notin \Sigma(x,p)$, we have $|xp| < |xq|$. The half spaces h_{pq} and h_{qp} are defined by the perpendicular bisecting plane of $\overline{pq}$, so all points $y \in S$ with $|yp| < |yq|$ are contained in h_{pq}. (A similar argument can be made for those points in h_{qp}.) Thus, x is the farthest point from p among those in $S \cap h_{pq}$, which is precisely the definition for the q-farthest point. □

Then, the approach presented in [4] for the planar case can be extended to $\mathbb{R}^3$. Specifically, for a fixed point $p \in S$, we can label all points $q \in S \setminus \{p\}$ with the q-farthest point f_{pq} as follows. First, sort S in order of non-increasing distance from p. Second, set f_{pq} for all points in S to be NULL. Third, pass through the sorted array and for each point q_i, in order, set f_{pq} to q_i for all points $q \in S$ that are not contained by the ball $\Sigma(q_i,p)$ and for which f_{pq} is set to NULL. That is, all points of S that are in $\cap_{k=1}^{i-1}\Sigma(q_k,p)$ but not in $\Sigma(q_i,p)$, are labeled with q_i, where $i = 1, 2, \ldots, n-1$.

After the sorting above, the last value in the sorted array of points in $S \setminus p$ is the point which has minimum Euclidean distance from p. Therefore $D(q_n,p) \equiv D(p,p) = 0$. This implies that f_{pq} is set for all points in $S \setminus p$. The sorted ordering also ensures that at any step in the algorithm, f_{pq} for any point q_i is the point corresponding to the smallest index j for which $q_i \in \cap_{k=1}^{j-1}\Sigma(q_k,p)$ and $q_i \notin \Sigma(q_j,p)$. This implies that the generic algorithm for finding the q-farthest point for a fixed point p and all $q \in S$ described in [4] for the planar case can also be applied in $\mathbb{R}^3$. We present this algorithm below and then show how to perform the computations associated with it efficiently in $\mathbb{R}^3$, so that by applying it for each $p \in S$ we achieve the claimed time and space bounds.

Algorithm description. Without loss of generality, assume that $n = 2^k$ for some integer k. Build a complete binary tree T with k levels as follows. The leaves of T are associated with the balls $\Sigma(q_i,p)$, $i = 1, 2, \ldots, n$, in order. That is, the leftmost leaf of T stores $\Sigma(q_1,p)$ and the rightmost leaf of T stores $\Sigma(q_n,p)$. Each internal node v of T stores a data structure associated with the common intersection of the balls that are leaf descendants of the subtree of T rooted at v. Given a point q, to find the smallest index j for which $q \in \cap_{k=1}^{j-1}\Sigma(q_k,p)$ and $q \notin \Sigma(q_j,p)$ start at the root of T and follow a path to a leaf of T, at each node v along the path performing the following test: if q is in the common intersection stored at the left child of v then go to the right child of v, else go to the left child of v. Clearly, the index associated with the leaf where this search ends corresponds to the sought j.

While the common intersection of n balls all having the same radius has complexity $O(n)$ [5], in our case the radii are not equal, and it is known that if the radii are not equal the common intersection can have complexity $\Omega(n^2)$. Thus, it is easy to check that a direct application of the algorithm above, with no other properties (like equal radii) in place, for each $p \in S$, would result in a solution for the MSST that takes cubic time and uses quadratic space, which is no better than brute force.

The astute reader may have noticed that answering whether a point q is inside the common intersection of a set of balls in $\mathbb{R}^d$ may not require the actual computation of the common intersection of the balls. In fact, a ray shooting based approach to answer this query has been presented in [1], for solving a related problem termed *off-line ball inclusion testing*. They use a standard geometric mapping, that lifts the point q to a paraboloid in dimension $d+1$ and maps the balls into $(d+1)$-dimensional hyperplanes. The intersections of the hyperplanes with the paraboloid, projected back to dimension d, are the original balls. With this lifting, answering whether a point q is inside the common intersection of n balls in $\mathbb{R}^d$ is equivalent to answering whether a point in dimension $d+1$ is below the lower envelope of a set of n $(d+1)$-dimensional hyperplanes. They [1] showed that using a static data structure for ray shooting queries, that allows for trade-offs between the preprocessing time and the query time, answering the latter question for a set of n query points can be done in time and space $O(n^{2-2/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$. Since we have to do this once for each $p \in S$, the overall time to find the MSST is $O(n^{3-2/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$. Each ray shooting data structure can be discarded after serving its purpose, so the overall space requirement remains $O(n^2)$. Thus, for any constant dimension d, we have:

Lemma 2 *Given a set S of n points in $\mathbb{R}^d$, $d \geq 2$ a constant, the MSST of S can be found in $O(n^{3-2/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$ time and $O(n^2)$ space.*

For $d = 3$ or 4, this gives an algorithm for the MSST with running time $O(n^{7/3} \log^{O(1)} n)$, which is better than the $O(n^{2.5+\epsilon})$ time algorithm in [4].

Surprisingly however, a faster solution can be obtained in $\mathbb{R}^3$ by actually computing the common intersection of the balls stored at internal nodes of T. For this, we need the following property.

Lemma 3 *Consider the common intersection of a set B of n balls in $\mathbb{R}^3$, all tangent to a point p. Then each ball can contribute at most one connected component to the boundary of the common intersection.*

Proof. Let a, b be two points on the boundary of the common intersection $bd(B^\cap)$ of the balls in B, both on the same bounding sphere s of some ball in B. The plane defined by a, b, and p intersects s in a circle c. Then the geodesic connecting a and b along c on s (an arc $\widehat{ab}$ of c) must be in $bd(B^\cap)$; otherwise, if another ball contains a and b but not some other point on $\widehat{pq}$, then the bounding sphere s' of that ball defines a circle c' in the plane of a, b, and p that has radius greater than that of c and contains p, a contradiction to the fact that s' is tangent to p. Thus, $bd(B^\cap) \cap s$ has at most one connected component. $\square$

Assuming general position (that is, no more than three bounding spheres intersect in a point excluding p) Lemma 3 implies the complexity of $bd(B^\cap)$ is $O(n)$.

Lemma 4 *Given a set S of n points in $\mathbb{R}^3$ and n balls $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, all tangent to a point p, there exists a data structure such that for each point $q \in S$, the smallest index i such that Σ_i does not contain q can be found in $O(\log^2 n)$ time. This data structure uses $O(n \log n)$ space and requires $O(n \log^2 n)$ preprocessing time.*

Proof. The data structure is the complete binary tree T described earlier enhanced with point location capability at each internal node. The intersection of the balls associated with the internal nodes is computed in a bottom-up fashion, using the algorithm in [7]. Although that algorithm was designed for equal radius balls, we note that the only place in that algorithm where equal radii plays a role is in obtaining the property that each ball contributes only one connected com-

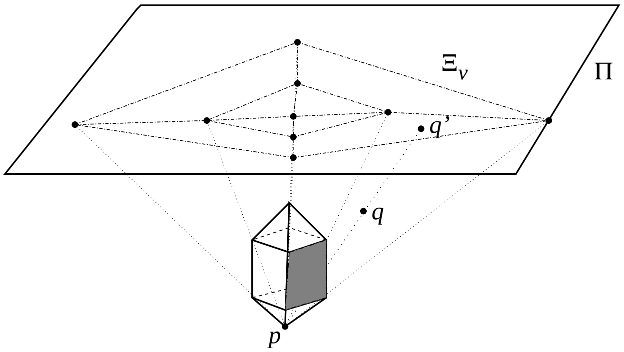

Figure 2: Illustrating the process of determining if a query point q is contained in $bd(B_v^\cap)$.

ponent to $bd(B^\cap)$. The algorithm computes the common intersection at each internal node by *merging* the intersections stored at its children and takes $O(n\log^2 n)$ time over T. Let Π' be the plane through p and tangent to $bd(B^\cap)$. At each internal node v, we *unfold* the common intersection by projecting it from p to a plane Π that is parallel to Π' and such that $bd(B^\cap)$ is sandwiched by Π and Π' (see Fig. 2). This unfolding can be done in time linear in the complexity of the intersection stored at v. We will refer to the resulting planar subdivision as Ξ_v. Finally, we preprocess Ξ_v for planar point location queries [3]. The overall construction time and space for T is dominated by the computation of the common intersection of balls. Thus, the data structure can be built in $O(n\log^2 n)$ time and uses $O(n\log n)$ space.

As explained earlier, a query with a point q follows a path from the root to a leaf of T, where the leaf gives the sought index. To decide whether q is inside the common intersection $bd(B_v^\cap)$ stored at an internal node v, we shoot a ray from p through q: if the ray does not intersect Ξ_v then $q \notin bd(B_v^\cap)$ else we obtain a point q' on Ξ_v (see Fig. 2). We perform a point location query for q', which takes $O(\log n)$ time [3]. If q' does not fall within a bounded face of Ξ_v, then the search at this node is done, and we traverse the left sub-tree. If q' is contained within some bounded face of Ξ_v, we check whether q is inside the corresponding ball. If not, we traverse the left sub-tree, otherwise we traverse the right sub-tree. The overall query time along the root-to-leaf path is thus $O(\log^2 n)$. □

Since the data structure for p can be discarded after f_{pq} is found for each $q \in S \setminus \{p\}$, we obtain:

Theorem 5 *Given a set S of n points in $\mathbb{R}^3$, the MSST of S can be found in $O(n^2\log^2 n)$ time with $O(n^2)$ space.*

3 Conclusion

In this paper we presented an algorithm for solving the MSST problem in $\mathbb{R}^3$ in $O(n^2\log^2 n)$ time using $O(n^2)$ space, almost matching the best known results for the planar case and improving the previously known results for $\mathbb{R}^3$. To achieve this, we proved an interesting result related to the complexity of the common intersection of n balls in $\mathbb{R}^3$, of possible different radii, that are all tangent to a given point p.

References

[1] G. Barequet, D. Z. Chen, O. Daescu, M. T. Goodrich, and J. Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. *Algorithmica*, 33(2):150–167, 2002.

[2] T.M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM J. on Computing*, 32(3):700–716, 2003.

[3] H. Edelsbrunner, L. J. Gubas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. of Computing*, 15(2):317–340, 1986.

[4] J. Gudmundsson, H. Haverkort, S.-M. Park, C.-S. Shin, and A. Wolff. Facility location and the geometric minimum-diameter spanning tree. *Lecture Notes in Computer Science*, 2462:146–160, 2002.

[5] A. Hepes. Beweis einer Vermutung von A. Vazsonyi. *Acta Math. Acad. Sci. Hungar.*, 7:463–466, 1956.

[6] J.-M. Ho, D.T. Lee, C.-H. Chang, and C.K. Wong. Minimum diameter spanning trees and related problems. *SIAM J. on Computing*, 20(5):987–997, 1991.

[7] E. A. Ramos. Intersection of unit-balls and diameter of a point set in $\mathbb{R}^3$. *Comput. Geom.*, 8:57–65, 1997.